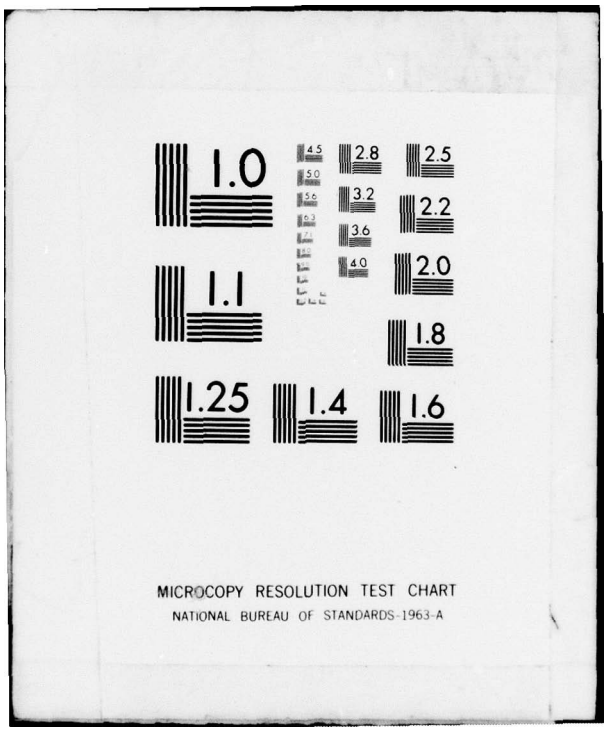
AD-A080 626 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS REMOTE DATA ENTRY SYSTEM. VOLUME II. CLUSTERING ADDITIO--ETC(U)
OCT 79 J D DYAR F30602-77-C-0174
UNCLASSIFIED PAR-79-53 RADC-TR-79-265-VOL-2 NL

1 OF 2
AD
A080626




1.0
4.5
5.0
5.6
6.3
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
1.1
1.25
1.4
1.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD A080626

(12)

LEVEL II

A080625

RADC-TR-79-265, Vol II (of two)
Final Technical Report
October 1979

MULTICS REMOTE DATA ENTRY SYSTEM
Clustering Additions to MOOS

Pattern Analysis & Recognition Corporation

Janet D. Dyar

DDC FILE COPY

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC
RECEIVED
FEB 14 1980
A

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

80 2 11 098

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-265, Vol II (of two) has been reviewed and is approved for publication.

APPROVED: Patricia J. Baskinger

PATRICIA J. BASKINGER
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISCP), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-265, Vol II (of two)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): MULTICS REMOTE DATA ENTRY SYSTEM, Volume II. Clustering Additions to MOOS.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report,
6. PERFORMING ORG. REPORT NUMBER: PAR-79-53	
7. AUTHOR(s): Janet D. Dyar	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0174
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Pattern Analysis and Recognition Corporation, 228 Liberty Plaza, Rome NY 13440	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F, 55971324, 13
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISCP), Griffiss AFB NY 13441	12. REPORT DATE: Oct 79
	13. NUMBER OF PAGES: 99
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Patricia J. Baskinger (ISCP)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Pattern Recognition
Pattern Analysis
Classification
Clustering
OLPARS
Data Entry

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report contains the user's manuals and software documentation for the Remote Data Entry System which is the front-end to the MULTICS Pattern Recognition Facility and the Cluster Analysis package which was added to MULTICS PLPARS. The Remote Data Entry System was designed to allow users of the MULTICS Pattern Recognition Facility the ability to input their data over the ARPANET from a Tektronix remote storage device. Once the data is input into the MULTICS System, routines are provided so that the user can easily restructure or cluster his database to perform different classification experiments.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

390 101

PREFACE

This is Volume II of the Final Report by Pattern Analysis and Recognition Corporation, 228 W. Dominick Street, Rome, New York written under Contract F30602-77-C-0174, for the Rome Air Development Center, Griffiss Air Force Base, New York. Mrs. Patricia J. Baskinger was the RADC Project Engineer. This report describes the clustering algorithms added to the MULTICS OLPARS Operating System under this effort.

Accession For	
NTIS GRA&I	☒
DDC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or special
A	

TABLE OF CONTENTS

Section		Page
1.	Introduction	1-1
1.1.	References	1-3
2.	Algorithm Description	2-1
2.1.	Distance Functions	2-1
2.2.	ISODATA - A Partitioning Algorithm	2-5
2.3.	MST - A Hierarchical Algorithm	2-9
3.	User's Manual	3-1
3.1.	General Remarks	3-1
3.2.	ISODATA	3-5
3.3.	MST	3-6
4.	Program Specifications	4-1
4.1.	Functional Overview	4-1

SECTION 1

INTRODUCTION

As its name implies, OLPARS (On-Line Pattern Analysis and Recognition System) combines graphic and statistical routines for the analysis of pattern recognition data. In the current version of MOOS (the OLPARS version under the MULTICS operating system), a clustering routine was not included among the repertoire of statistical procedures but would be most appropriate in a pattern recognition environment.

A study [18] was thus initiated to determine which clustering algorithm(s) would be most appropriate to add to OLPARS to complement the existing capabilities. It was important to select an algorithm which was not already represented in the MOOS system in some form to avoid duplication.

Algorithms considered were those which most successfully complemented the functioning of the MULTICS OLPARS system (MOOS), were consistent with the purposes of the MOOS system, were accurate, and had time and storage factors practical for data sets of the size typically input to MOOS.

The set of clustering algorithms that were considered appropriate for inclusion in MOOS can be classified as either hierarchical or partitioning. Hierarchical algorithms tend to be agglomerative in nature, starting (conceptually) with one cluster for each data element and sequentially combining the clusters until all the data elements are contained in a single cluster. Partitioning algorithms start with a specific number of clusters, typically defined a priori. During successive iterations, the number of clusters remains relatively constant while their membership changes. In reviewing the clustering literature, it became evident that neither approach was necessarily better. Rather, it was decided that two complementary algorithms, one from each of the major algorithm classes would be appropriate for inclusion in MOOS. The two algorithms are ISODATA and a variant of the minimal spanning tree (MST) approach.

This report contains detailed design and user-oriented information concerning the clustering functions ISODATA and MST. These two functions are additions to the currently implemented version of MOOS existing on the Honeywell 6180 MULTICS Computer System at RADC. Section 2 contains a description of each function and an overview of why the particular algorithm was chosen. Also included in Section 2 is a description of the distance functions available to the user. Section 3 is the User's Manual for the two added functions. It is assumed here that the user has a working knowledge of the basic system capabilities of MOOS. Section 4 contains file descriptions and program documentation.

1.1. REFERENCES

1. Anderberg, M.R. Cluster Analysis for Applications, New York: Academic Press, 1973.

2. Ball, G.H. A comparison of some cluster-seeking techniques, Report Number RADC-TR-66-514, Stanford Research Institute, Menlo Park, California, 1966. (AD643287)

3. Ball, G.H. Classification Analysis, Technical Note, Stanford Research Institute, Menlo Park, California, 1970. (AD716482)

4. Ball, G.H. & Hall, D.J. Some fundamental concepts and synthesis procedures for pattern recognition preprocessors, Paper presented at the International Conference on Microwaves, Circuit Theory, and Information Theory, Tokyo, Japan, Sept. 1964.

5. Ball, G.H. & Hall, D.J. ISODATA, A Novel Method of Data Analysis and Pattern Classification, Stanford Research Institute, Menlo Park, California, 1965. (AD699616)

6. Ball, G.H. & Hall, D.J. PROMENADE--An On-Line Pattern Recognition System, Report Number RADC-TR-67-310, Stanford Research Institute, Menlo Park, California, 1967. (AD822174)

7. Blashfield, R.K. Mixture model of cluster analysis: accuracy of four agglomerative hierarchical methods, Psychological Bulletin, 1976, 83, 377 - 388.

8. Cormack, R.M. A review of classification, Journal of the Royal Statistical Society, Section A, 1971, 134, 321 - 367.

9. Duda, R.O. & Hart, P.E. Pattern Classification and Scene Analysis. New York: Wiley, 1973.

10. Hall, D.J., Ball, G.H., Wolf, D.E., & Eusebio, J. PROMENADE: An Improved Interactive-Graphics Man/Machine System for Pattern Recognition, Report Number RADC-TT-68-572, Stanford Research Institute, Menlo Park, California, 1969. (AD692752)

11. Hartigan, J.A. Clustering Algorithms, New York: John Wiley, 1975.

12. Johnson, S.C. Hierarchical Clustering Schemes, Psychometrika, 1967, 32, 241 - 254.

13. Koopman, R.F. & Cooper, M. Some problems with Minkowski distance models in multidimensional scaling, Paper presented at the Psychometric Society Meeting, March 1974, Stanford, CA.

14. Mezzich, J.E. Comparative performance of cluster analytic methods, workshop on classifying cultural and social data, Charleston, South Carolina, March 23 - 27, 1977.

15. Sammon, J.M. On-Line Pattern Analysis and Recognition System (OLPARS), Report Number RADC-TR-68-263, Rome Air Development Center, USAF, 1968 (AD675212)

16. Sokal, R.R. & Sneath, P.H. Principles of Numerical Taxonomy, San Francisco: Freeman, 1963.

17. Zahn, C.T. Graph-theoretic methods for detecting and describing Gestalt clusters, IEEE Transactions on Computers, 1971, C-20, 68 - 86.

18. Hersh, H.M., Clustering Algorithm Evaluation for Pattern Analysis Applications, PAR Report No. 79-6, Submitted to RADC Jan 1979.

SECTION 2

ALGORITHM DESCRIPTION

2.1. DISTANCE FUNCTIONS

Clustering is typically performed by grouping the data elements together on the basis of the distance between the elements. How one defines distance is quite independent of the clustering algorithm to be used. Yet the particular distance function used may have a significant bearing on the ultimate clustering solution [1, 8, 11]. In this section, three of the distance functions are discussed. Based on what is currently available in MOOS and on what functions are most frequently used, these distance functions were selected to give the user several options by which distance may be defined.

2.1.1. Number of Dimensions

Sets of data are usually entered into the MOOS system as data vectors, where an element of a data vector represents the value of the item of data on that particular measure (dimension). A user may want to define the distance between two data vectors using the full data vectors or only some lower dimensional projection of the vectors. Using a subset of the measures may be a reasonable approach when several of the measures are significantly correlated, or when subsets of measures represent very different aspects of the data. In any case, it should be possible for a user to specify (optionally) what subset of measures should be used to determine the inter-element distances.

2.1.2. Euclidean Distance

Perhaps the most familiar distance metric is the Euclidean distance between two data vectors, X_i, X_j:

$$d_{ij} = \{ \sum_{k=1}^{N} (x_{ik} - x_{jk})^2 \}^{1/2}$$

where

x_{ik} = value of data vector X_i on measurement k,

and

N = number of measures.

A generalization of the Euclidean distance, often used in multidimensional scaling [e.g., 13] as well as in clustering, is the Minkowski distance function

$$d_{ij} = \{ \sum_{k=1}^{N} | x_{ik} - x_{jk} |^c \}^{1/c}$$

where

$c > 0$

When c=2, this reduces to the Euclidean distance. For c=1, the result is the "city-block" metric.

In a recent evaluation of cluster analytic methods [14], the differences in performance between various clustering algorithms were much greater than the within differences due to using either the Euclidean or city-block metric; in fact, differences due to the metric were minimal. Because of this, result, and because there were no compelling reasons why a generalized Minkowski distance function should be made available, only c=2 (Euclidean distance) was implemented.

2.1.3. Weighted Distance

Quite often the dimensions of a data vector will represent measurements from significantly different scales (e.g., milliseconds and kilometers). In such cases, an absolute difference in scale values will have a differing meaning across the various measurements. One way of equating the dimensions to eliminate these differences is to normalize (standardize) each measurement by dividing the corresponding elements in the data vectors by the standard deviation of the measurement. This normalization procedure is equivalent to the normxfrm function in MOOS (and to the inverse variance weighting found in the nearest mean vector function).

The computational form of the metric used in this option is

$$d_i = \sum_{j=1}^{l} \frac{(x_j - \mu_j^i)^2}{v_j^i}$$

where

x = unknown vector = $x_1, x_2, \ldots x_l)$

μ_i = mean vector of class i = $(\mu_1^i, \mu_2^i, \ldots \mu_l^i$

and

v_j^i = variance of the jth component of the ith class.

2.1.4. Mahalanobis Distance

When there is a possibility that the various measurements comprising the data vectors are correlated, one might consider using the Mahalanobis distance to define the proximity of two data vectors, x_i and x_j:

$$d_{ij} = (x_i - x_j)V^{-1}(x_i - x_j)^t$$

where $\underline{V}^{-1}$ is the inverse of the covariance matrix for the measures. When the set of measures are all uncorrelated, $\underline{V}$ reduces to a diagonal matrix of variances, and the result is the normalized Euclidean distance. When the measures are correlated, this distance function will, in addition, compensate for the redundant information by orthogonalizing the space in which the data vectors reside.

2.2. ISODATA - A PARTITIONING ALGORITHM

In examining the various partitioning schemes, the ISODATA routine was evaluated positively on the performance criteria (see, for example [14]), and was chosen as the most appropriate partitioning algorithm. The algorithm is more comprehensive than many of the other routines examined. In fact, several of the other clustering procedures can be considered special cases of the ISODATA routine. Moreover, it is an algorithm that has been extensively tested and improved [2-6, 10 ,14], is well-known in the pattern recognition community, and was even implemented in an early version of OLPARS, under a contract with RADC [15]. It is an algorithm that is suitable for an interactive environment as provided by MOOS where a user can evaluate and act on intermediate results between iterations.

The version of ISODATA implemented combines features from several different versions, and is discussed in Anderberg [1]. The main algorithm consists of successive iterations of obtaining seed points (cluster centroids) for the clusters and assigning each data element to the cluster with the nearest centroid. After every data element is assigned to a cluster, the centroids are computed and the processes are repeated. The calculation/allocation cycle continues until convergence or until the maximum number of iterations has been exceeded. Clusters may also be split apart or merged, based on several criteria. The specific algorithm is described below:

1. A set of parameters is defined explicitly. The parameters are:

MAXCY - Maximum number of calculation/allocation cycles

NSPLIT - Splitting/Merging criterion

MAXMER - Maximum number of merges per iteration

THETAC - Maximum intercentroid distance for merging

THETAE - Splitting parameter

MINCL - Minimum cluster size

ITERMAX - Maximum number of iterative cycles

inc - Starting number of clusters to be found

d - Maximum centroid distance for initial cluster assignments

The meanings of these parameters will become clear from the description of the algorithm.

2. Select the initial seed points. These are generated by the system. The generation method is the one suggested by Ball and Hall [5], where the overall mean vector of the data set is taken as the first seed point. The data elements are then read in one at a time. If a data element is more than some distance, d, from all previously chosen seed points, then it will be chosen as an additional seed point. The process continues until some number, inc, of seed points are chosen. This procedure is simple enough that it can be repeated several times if the seed point sets are not adequate, either because too few seed points were chosen, or because not enough of the data set was examined.

3. Assign each data element to the cluster with the nearest centroid.

4. Calculate new centroids based on the means of the data elements in each of the clusters. Steps 3 and 4 are repeated until no data element changes cluster membership or the number of iterations exceeds MAXCY.

5. Delete any clusters containing less than MINCL data elements. The elements in these clusters will not be considered further, but will be put on a list of outliers for future reference.

6. Attempt to either split or merge clusters according to the following conditions:

   a. Attempt to merge clusters if the number of clusters is greater than twice NSPLIT.

   b. Attempt to split clusters if the number of clusters is less than one half NSPLIT.

   c. Otherwise alternate merging and splitting on even and odd iterations, respectively.

7. Repeat steps 3 and 4 up to MAXCY times or until convergence.

8. Repeat steps 5, 6 and 7 up to ITERMAX times or until the process converges.

For a merging operation, all intercentroid distances are computed. If the distance between the two nearest centroids is less than THETAC, then the two clusters are merged. This process is repeated up to MAXMER times.

For a splitting operation, a cluster is tentatively chosen for splitting if the within cluster standard deviation for any dimension exceeds the product of THETAE and the standard deviation of that dimension in the original data set. In this case, two clusters are formed, with data elements assigned according to whether they are above or below the mean on that dimension. The

centroids of these two new clusters are computed, and if the distance between them is at least 1.1 times THETAE, then the split is accepted.

As is pointed out by Anderberg [1], the effects of this splitting operation is to constrain the cluster widths in every direction; it is highly unlikely that elongated clusters will be found. This potential shortcoming will be addressed by the hierarchical clustering algorithm, which was intentionally chosen to complement the ISODATA routine. The hierarchical algorithm will be discussed in the next section.

2.3. MST - A HIERARCHICAL ALGORITHM

Most of the hierarchical clustering routines require the calculation and storage of the matrix of all pairwise distances among the data vectors. As the data set becomes large, this distance matrix can become much larger than the original data set. As a result, most hierarchical clustering algorithms are not practical for pattern recognition applications where relatively large data sets are often encountered. However, there is one algorithm which can operate on large data sets and is complementary to the ISODATA algorithm, the single linkage algorithm, also known as the minimal spanning tree approach [1, 12, 9, 11, 12, 15, 17]. Although this algorithm is not one of the more robust procedures in any general sense [7, 14], it does have the unique ability to detect elongated clusters, an ability lacking in ISODATA and in most other clustering routines.

In general, single linkage clustering is also inefficient for large data sets; however, Anderberg has developed a novel approach in which the distance matrix search and evaluation procedures are replaced by a partition-exchange sort on the distance matrix [1]. The algorithm is rather simple and is outlined below:

1. The distances between all pairs of data elements are calculated and output to a system storage device (e.g., disk file or tape) as triples consisting of the two data element identifiers and the value of the distance between them. For N data elements, there will be N(N-1)/2 of these triples.

2. A partition-exchange sort routine is used to sort the triples on the distance field in ascending order.

3. The first link is formed between the pair of data elements closest together: i.e., the first triple in the sorted list.

4. Each triple is then read in order and one of the following actions occur:

   a. If the two data elements in the triple are not already connected through one or more links, then link these two data elements.

b. If the two data elements are already linked together, then ignore the triple and read in the next one.

5. Continue reading in the triples from the sorted list until N-1 links have been formed. At this point, all N data elements will be contained in this minimal spanning tree.

For small data sets, one can often look at the resulting tree structure to evaluate the clusters inherent in the data set. However, for data sets of even moderate size (N > >100), there is too much information in the tree to examine visually. Thus, a postprocessing operation is needed in which the clusters are made explicit. According to Zahn [17], the basic idea is to detect inherent separations in the data by deleting links from the minimal spanning tree which are significantly longer than nearby links. Such a link is called inconsistent. Operationally, a link will be considered inconsistent if its length (i.e., the distance between the two data elements) is more than K times the mean of the length of the nearby links. (For most applications, K=2 would appear to be a reasonable choice.) For purposes of implementation, nearby will be defined as any link containing either of the data elements in the link under evaluation.

Although by varying the strictness of the criteria for defining an inconsistent link, it is possible to influence the ultimate number of clusters detected, the user is not really able to specify the number of clusters desired a priori. Unlike the partitioning algorithms, the hierarchical clustering algorithm is influenced much more by the inherent structure of the

data then by the interests and bias of the user. Thus, not only does it complement the ISODATA algorithm relative to the types of clusters it detects best, but it is also complementary relative to the amount of influence that the user can bring to bear on the analysis.

SECTION 3

USER'S MANUAL

3.1. GENERAL REMARKS

This manual contains descriptions of the two user functions, ISODATA and MST. It is designed to provide a potential user of the MOOS system with sufficient information to allow functional utilization of the two functions.

The standard terminal from which MOOS commands are executed is the Tektronix 4014 storage tube display interfaced to the Honeywell 6180 MULTICS processor. The MULTICS control language has been utilized to the fullest possible extent in the development of MOOS, and therefore a working knowledge of the MULTICS environment is essential. For further information, consult the MULTICS Programming Manual. The user function calls are input via the console keyboard and consist of a simple program name followed by any required or optional parameters. Dialogue concerning additional information required for program operation is handled by standard terminal input/output operations as specified within this manual.

Initiation of the MOOS Environment

Entrance into the MOOS program environment can be accomplished by a MULTICS user via the execution of the command hello-moos. Upon completion, this function provides the user with an orientation to the standard system display. The hello-moos command is not a MULTICS function and may be

utilized only by users for whom linkage to the MOOS directories has already been provided.

The Standard Data Set Selection Parameters

Each of the two user programs may be called by the user with up to two optional parameters, which represent the tree name and the name of the data class upon which the operation is to be performed. Rules for these parameters follow:

- All tree names are required to be five to eight characters in length. Data class names are four characters in length (the final character is used as a display symbol).

- If two parameters are input, the first will represent the tree name. The tree name input must currently exist within the system. The second parameter must be a legal data class name within the selected data tree.

- If one parameter is input, it may represent either a tree name or a data class name. If it is five to eight characters in length, it is taken to be a tree name, and the current data class is set by default to the senior node (symbolized by "****"). A four-character input is assumed to be a data class name within the current data tree.

- If no parameters are input, the current data tree and class parameters are not changed and operations of the called function are executed upon the same data set as the previous option.

"Cluster File" Format

Each cluster file created by isodata and mst, contains specific information regarding the number of vectors in the cluster, the data vectors, the mean of the cluster and either the variance or inverse covariance of the cluster, depending on the type of distance metric used. The format of the file is as follows (See Figure 3-1):

where

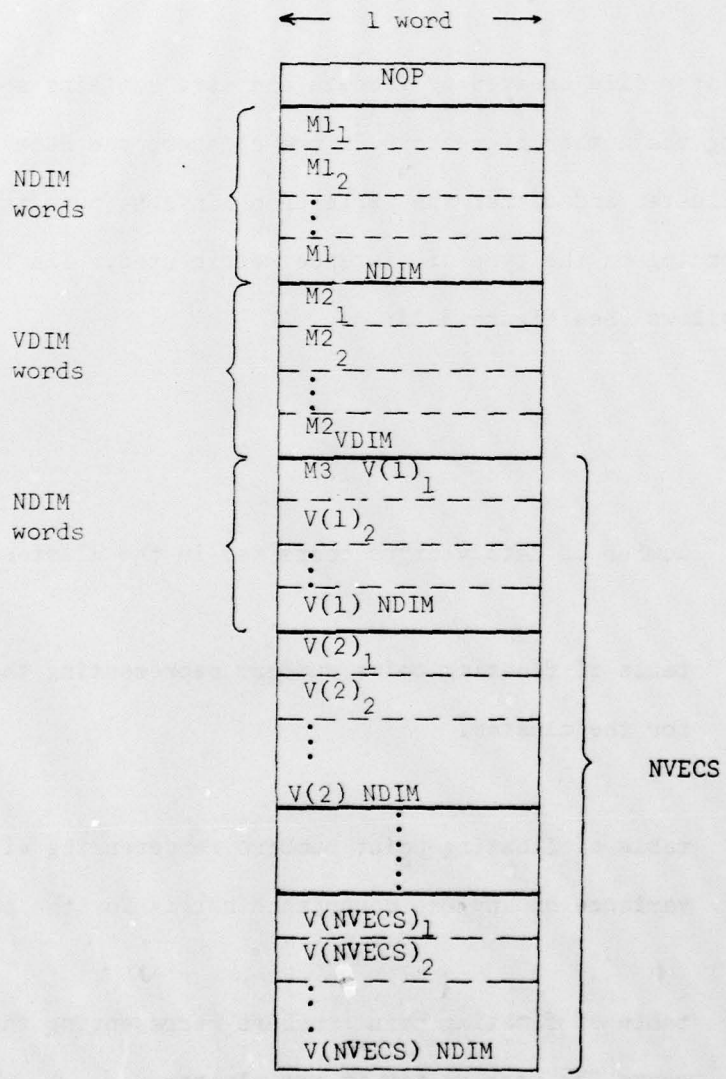
NOP - number of data vectors contained in the cluster file.

M1 - table of floating point numbers representing the mean vector for the cluster.

M2 - table of floating point numbers representing either the variance or inverse covariance matrix for the cluster.

M3 - table of floating point numbers representing the data measurements for each vector in the cluster.

Figure 3-1 Cluster File Format



ndim - number of dimensions in the data vectors within the cluster. The dimension value is obtained from the data class file.

vdim - number of dimensions in the variance or inverse covariance matrix in the cluster. The value vdim is determined by the distance metric used and the number of dimensions in the data vectors. If using the weight vector metric, "w", then vdim= ndim. If using the mahalanobis metric, then vdim=ndim* (ndim +1)1/2.

3.2. ISODATA

Once in the MOOS directory, isodata is initiated by the call

isodata [(treename)][(nodename)]

with the parameters being input as mentioned earlier in Section 3. The data set is examined to verify that it is a lowest node. If not, isodata will notify the user that it can only operate on the lowest node and will then terminate. Otherwise, isodata calls other subroutines to determine the clusters of the data set.

Once the clusters have been created, if the user has indicated that he does not want a tree created from the clusters, then isodata will query the user about saving the clusters in the process directory. If the user wants the files saved, then all cluster files are renamed. If the user does not

want the files saved, then all cluster files are deleted. The deletion or renaming of the cluster files is to enable the user to call upon isodata again, in the same process without destroying any existing cluster files.

If the user has indicated that he wants a tree created from the clusters, then isodata will query the user for the new tree name. Treeiput is then called to create the tree. Isodata terminates by calling "option" thus preparing the user for his next MOOS option.

3.3. MST

Within the MOOS directory, mst is initiated by the call

```
mst [(treename)] [(nodename)]
```

with the parameters being input as mentioned in "General Remarks" of Section 3. The data set is examined to verify that it is a lowest node. If not a lowest node, then mst will notify the user that operations can only be done on lowest nodes and mst will then terminate. Otherwise, mst links all the data points in the data set together as one cluster. Then, based upon the user-supplied values for a maximum and minimum link length, links are deleted if they do not fall between the maximum and minimum link length. The user is notified of the number of clusters formed as a result of deleting links. He will then be asked if he wants to change the maximum and minimum values for

the link length. If he does, then the new values are obtained from the user, mst deletes all clusters formed, links all points of the data set together into 1 cluster and repeats the cycle of deleting links.

If the user does not want to change the link length values and has specified that he does not want a tree created from the clusters, the following occurs. The user is asked if he wants the cluster files saved in the process directory. If so, then all cluster files are renamed. If he does not want the files saved, then all cluster files are deleted. The deletion or renaming of the cluster files is to enable the user to call upon mst again, in the same process, without destroying any existing cluster files.

If the user has indicated he wants a tree created from the clusters, then mst queries the user for the new tree name and calls upon treeiput to create the tree. Mst terminates by calling "option" thus preparing the user for his next MOOS option.

SECTION 4

PROGRAM SPECIFICATIONS

4.1. FUNCTIONAL OVERVIEW

This section consists of a description of the functional organization of the two added features, ISODATA and MST. It is not aimed toward the average MOOS user, but is for the system programmers who might be interested in modifying or expanding the MOOS system.

The following pages describe the programs added to the existing MOOS system. Each program description includes a complete description of input and output parameters and file settings, a functional description of the program and an algorithm of the program. Internal subroutines are transparent to the user and cannot be called from the console directory but are called by the MOOS functions.

Table 4-1 is a listing of the two MOOS functions added to the MOOS system. Table 4-2 is an alphabetic listing and page number index of all routines used.

Table 4-1 Numerical Index of Added MOOS Functions

Number	Functions
217	isodata
218	mst

Table 4-2 Alphabetic Listing and Page Number Index of all Routines

Name	Description	Page
aspts	assigns data points to clusters	4-7
cdis	calculates distance between two clusters.	4-10
clearpts	zeros out all data points in all clusters.	4-11
compcent	calculates mean of each cluster.	4-13
covc	calculates the variance and inverse covariance matrices of each cluster.	4-15
ctrdata	creates the file "treedata" from the clusters.	4-17
dcen	deletes all clusters having too few data points.	4-19
dis	calculates the distance between a vector of the data set and a cluster, using the covariance matrix of the original data set.	4-20

Table 4-2 Alphabetic Listing and Page Number Index of all Routines

(Continued)

Name	Description	Page
disl	calculates the distance between a vector of the data set and a cluster, using the inverse covariance matrix of the cluster.	4-23
dlink	deletes all inconsistent links in the clusters.	4-25
flinks	computes links between all vectors of the data set.	4-27
getparams	queries the user for all parameters needed in isodata.	4-30
iclc	generates the clusters.	4-33
init	obtains all parameters from the user for mst.	4-37
isodata	determines clusters existing in the data set.	4-39

Table 4-2 Alphabetic Listing and Page Number Index of all Routines

(Continued)

Name	Description	Page
maxdis	determines which cluster a vector is farthest from.	4-43
merger	attempts to merge two clusters into one.	4-46
mindis	determines which cluster a vector is closest to.	4-48
mindisl	determines which cluster a vector is closest to.	4-50
mst	detects elongated clusters in a data set.	4-52
pcfile	puts vectors into cluster files and calculates mean of each cluster.	4-55
reaspts	reassigns data points to clusters.	4-56
recomp	calculates the mean of any cluster having data points which changed clusters.	4-58
sorty	uses a quick sort to sort an array on the distance element.	4-60

Table 4-2 Alphabetic Listing and Page Number Index of all Routines

(Continued)

Name	Description	Page
split	attempts to split one cluster into two.	4-63
trip	creates array of all pairs of data elements and the distance between each element of a pair.	4-65
vdis	calculates distance between two vectors.	4-67

Internal Subroutine Name: aspts

Calling Sequence: call aspts (tptr, dfptr, ccptr, aptr, tsptr, metflag, classn, itcnt)

Input Parameters:

tptr	-	pointer to treename file.
dfptr	-	pointer to data file.
ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each of the cluster files.
tsptr	-	pointer to array of temporary symbols which specify what cluster each vector is assigned to.
metflag	-	distance metric flag.
classn	-	number of class in treename file.
itcnt	-	number of iterations isodata has completed.

Program Description: aspts assigns the vectors of the data set to a cluster. This is accomplished by calling a distance routine, mindis, to determine which cluster the vector is closest to, and then placing the vector in that cluster. If the user has specified that he wants to see how isodata is progressing upon completion of assigning all vectors to a cluster, the user is notified of the number of vectors in each cluster.

Algorithm: See following page

```
Entry:                    aspts (tptr, dfptr, ccptr, aptr, tsptr,
                          metflag, classn, itcnt)

    begin aspts;

        if iteration count = 1 then examine all vectors;

            call mindis to determine which cluster the vector
                 is closest to;

            set temporary symbol = cluster number;

            get pointer to cluster file;

            increment number of points in file;

            add vector to file;
        end;
        else if iteration count ^= 1 then do;

            examine each vector;

            if temporary symbol ^= 0 then do;

                call mindis to determine which cluster the
                     vector is closest to;

                set temporary symbol = cluster number;

                get pointer to cluster file;

                increment number of points in file;

                add vector file;
            end;
        end;
         if list flag is on then print the number of points in each
            cluster;

    end aspts;
```

Internal Subroutine Name: cdis

Calling Sequence: call cdis (ccptr, aptr, s1, s2, metflag, dist)

Input Parameters:

ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each cluster file.
s1	-	cluster number of the first cluster.
s2	-	cluster number of the second cluster.
metflag	-	distance metric flag.

Output Parameters

dist	-	distance between the two clusters.

Program Description: cdis, using the distance metric specified by metflag, calculates the distance between the 2 clusters specified by s1 and s2 respectively.

Algorithm: See following page

```
Entr :                          cdis (ccptr, aptr, s1, s2, metflag, dist)
     begin cdis;
          get pointer to each cluster file;
          if metflag = "e" then
               use euclidean distance formula;
          else if metflag = "w" then use
               weighted vector formula;
          else if metflag = "m" use mahalanobis
               distance formula;
          calculate distance;
end cdis;
```

Internal Subroutine Name: clearpts

Calling Sequence: call clearpts (ccptr, aptr, tsptr, dfptr, z)

Input Parameters:

ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each of the cluster files.
tsptr	-	pointer to array of temporary symbols which specify what cluster each vector is assigned to.
dfptr	-	pointer to data file.
z	-	flag indicating whether or not to reset the temporary symbols of the vectors.

Program Description: clearpts clears out all points from each cluster file. Then if z = 2, it resets all negative temporary symbols.

Algorithm: See following page

Entry: clearpts (ccptr, aptr, tsptr, dfptr, z)

```
begin clearpts;

    do 1 to number of clusters;

        get pointer to cluster file;

        set all points to zero;

        set number of points to zero;

    end;

    if z = 2 then do;

        examine all vectors;

        if temporary symbol is negative then reset to a positive
        number;

    end;

end clearpts;
```

Internal Subroutine Name: compcent

Calling Sequence: call compcent (dfptr, ccptr, aptr)

Input Parameters:

dfptr	-	pointer to data file.
ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each of the cluster files.

Program Description: compcent calculates the mean of each cluster.

Algorithm: See following page

```
Entry:                          compcent (dfptr, ccptr, aptr)
    begin compcent;
        do 1 to number of clusters;
        get pointer to cluster file;
        initialize mean to zero;
        calculate mean;
        end;
    end compcent;
```

Internal Subroutine Name: covc

Calling Sequence: call covc (ccptr, aptr, metflag)

Input Parameters:

ccptr - pointer to cluster file.

aptr - pointer to array containing a pointer to each of the cluster files.

metflag - distance metric flag.

Program Description: covc calculates the variance or inverse covariance matrix for each cluster. The metflag determines which matrix covc needs to calculate. If the metflag is "w" (for the weighted vectors) then the variance matrix is calculated. If the metflag = "m" (for the mahalanobis distance) then the inverse covariance matrix is calculated.

Algorithm: See following page

```
Entry:                          covc (ccptr, aptr, metflag)

     begin covc;

          if metflag = "w" for weighted distance do;

               do for each cluster;

                    get a pointer to cluster file;

                    initialize variables;

                    calculate variance;

               end;

     end;

      else if metflag = "m" for mahalanobis distance do;

          do for each cluster;

               get pointer to cluster file;

               initialize variables;

               compute upper triangle of covariance matrix;

          do expand packed covariance matrix to full matrix;

               create an ic region for inverse matrix routine;

               call invertmat to invert covariance matrix;

               store only upper triangle of full inverse covariance
                    matrix;

          end;

        end;

end covc;
```

Internal Subroutine Name: ctrdata

Calling Sequence: call ctrdata (ccptr, dfptr, aptr, tdptr)

Input Parameters:

ccptr	-	pointer to cluster file.
dfptr	-	pointer to data file.
aptr	-	pointer to array containing a pointer to each of the cluster files.
tdptr	-	pointer to the treedata file.

Output File Settings: creates the file treedata.

Program Description: ctrdata creates the file treedata for use in the subroutine treeiput, which creates a MOOS tree from the treedata file. This routine is used when the user wants a MOOS tree created from the clusters.

Algorithm: See following page.

```
Entry:                          ctrdata(ccptr, dfptr, aptr, tdptr)
     begin ctrdata;
          initialize tvecs to zero;
     do for each cluster;
          get pointer to cluster file;
          sum the number of points in cluster file and tvecs;
     end;
     create treedata file;
     do for each cluster;
          name each cluster and put in treedata file;
          put number of points in cluster in treedata file;
     end;
     do for each cluster;
          put points of cluster in treedata file;
          put vector id in treedata file;
     end;
end ctrdata;
```

Internal Subroutine Name: dcen

Calling Sequence: call dcen (ccptr, aptr, tsptr, mincl, nodl)

Input Parameters:

ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each cluster file.
tsptr	-	pointer to array of temporary symbols which specify what cluster each vector is assigned to.
mincl	-	indicates the minimum number of vectors needed to constitute a cluster.

Output Parameters:

nodl	-	number of deleted clusters.

Program Description: dcen examines the number of vectors in each cluster. If the number of points in the cluster is less than the user-specified minimum cluster size (mincl), then the cluster is deleted. The temporary symbol of each vector in the deleted cluster is set to zero. If the user has indicated that he wants to see how isodata is progressing, then the number of deleted vectors is printed.

Algorithm: See following page

```
Entry:                          dcen (ccptr, aptr, tsptr, mincl, nodl)
     begin dcen;
          initialize number of deleted clusters to zero;
          get process directory;
          do i = 1 to number of clusters;
               get pointer to cluster file;
               if number of points < maximum cluster size, then do;
                    do j = 1 to number of vectors;
                         if temporary symbol = i, then reset temporary
                         symbol to zero;
               end;
               get cluster file name;
               delete cluster file;
               rename all cluster files;
               turn decrement switch on;
               end;
               if decrement switch on, then do;
                    decrement i;
                    turn decrement switch off;
               end;
          end;
               if lflag, then print the number of deleted clusters;
end dcen;
```

Internal Subroutine Name: dis

Calling Sequence: call dis (tptr, ccptr, dfptr, vn, classn, metflag, dist)

Input Parameters:

tptr	-	pointer to treename file.
ccptr	-	pointer to cluster file.
afptr	-	pointer to data file.
vn	-	element number of vector in data file.
classn	-	number of data class in treename file.
metflag	-	distance metric flag.

Output Parameters:

dist	-	distance between the vector specified by vn and the cluster pointed to by ccptr.

Program Description: dis, using the distance metric specified by metflag, calculates the distance between the vector specified by vn and the cluster pointed to by ccptr.

Algorithm: See following page

Entry: dis (tptr, ccptr, dfptr, vn, classn, metflag, dist)

```
begin dis;

     if metflag = "e" then use euclidean distance formula;

     else if metflag = "w" then use weighted vector formula;

     else if metflag = "m" then use mahalanobis distance formula;
     calculate distance;

end dis;
```

Internal Subroutine Name: disl

Calling Sequence: call disl (ccptr, dfptr, vn, metflag, dist)

Input Parameters:

ccptr	-	pointer to cluster file.
dfptr	-	pointer to data file.
vn	-	number of vector in data file.
metflag	-	metric distance flag.

Output Parameters:

dist	-	distance between the vector specified by vn and the cluster specified by ccptr.

Program Description: disl, using the appropriate distance formula specified by metflag, calculates the distance between the vector, vn, and the cluster pointed to by ccptr.

Algorithm: See following page.

```
Entry:                          disl (ccptr, dfptr, vn, metflag, dist)

     begin disl;

          if metflag = "e" then use euclidean distance formula;

          else if metflag = "w" then use weighted vector formula;

          else if metflag = "m" then use mahalanobis distance formula;

          calculate distance;

end disl;
```

Internal Subroutine Name: dlink

Calling Sequence: call dlink (lmax, lmin, trptr, pptr, cptr, theta, ccptr, aptr)

Input Parameters:

lmax	-	maximum link length.
lmin	-	minimum link length.
trptr	-	pointer to tarray.
pptr	-	pointer to p array.
cptr	-	pointer to c array.
theta	-	variable used in determining inconsistent links.
ccptr	-	pointer to cluster file.
aptr	-	pointer to array of pointers to cluster files.

Output File Settings: creates cluster files.

Program Description: dlink is used to delete any inconsistent links in the cluster as determined by lmax, lmin, and theta, and to create the files for each cluster found.

Algorithm:

Entry: dlink(lmax, lmin, trptr, pptr, cptr, theta, ccptr, aptr)

```
begin dlink;
    number of clusters = 1;
    i = number of vectors - 1;
    do loop;
    if distance between 2 points is < lmax and > lmin then link is okay;
    else do;
        create new cluster file;
```

```
            reset value in c array;
            determine mean of nearby links;
            if distance > theta * mean, then do;
                create new cluster file;
                reset value in c array;
            end;
        end;
        decrement i;
        if i = 1, quit loop;
end;
end dlink;
```

Internal Subroutine Name: flinks

Calling Sequence: call flinks (trptr, cptr, pptr, metflag)

Input Parameters:

trptr	-	pointer to tarray.
cptr	-	pointer to c array.
pptr	-	pointer to p array.
metflag	-	distance metric flag.

Program Description: flinks determines the links in the data set while, at the same time creating c array and the p array. The c array is in 1-1 correspondence with the data file and indicates which cluster the vector is in. The p array is used as an index back into the tarray.

Algorithm: See following page.

```
Entry:                          flinks (trptr, cptr, pptr, metflag)
    begin flinks;
        initialize index i to zero;
        initialize index l to one;
            do where l < number of vectors;
                increment i by 1;
                    K1 = value of c array of tarray (i), vec1;
                    K2 = value of c array of tarray (i), vec2;
            if K1=K2 then do;
                if K1=0 then do;
                    set c (K1) = l;
                    set c (K2) = l;
                    set p (l) = i;
                     increment l by 1;
            end;
        end;
         else if K1=0 then do;
            set c(K1)=K2;
                set p(l) = i,
                increment l by 1;
            end;
             else if K2=0 then do;
                set c (K2)=K1;
                set p (l) = i;
                increment l by 1;
            end;
```

```
      else do;
     do i=1 to number of vectors;
          if c(j) = K2 then c(j) = K1;
end;
 set p(1)=i;
 increment 1 by 1;
end;
end;
number of clusters = 1;
end flinks;
```

Internal Subroutine Name: getparams

Calling Sequence: call getparams (metflag, maxcy, nsplit, maxmer, thetac, thetae, mincl, itermax, d, lflag, tflag)

Output Parameters:

metflag	-	distance metric flag
maxcy	-	maximum number of calculation-allocation cycles.
nsplit	-	used in determining whether to split or merge clusters.
maxmer	-	maximum number of merges per iteration.
thetac	-	maximum intercentroid distance for merging clusters.
thetae	-	used in determining whether a split is acceptable or not.
mincl	-	minimum number of points which will keep a cluster from being deleted.
itermax	-	maximum number of iterations through isodata.
d	-	maximum centroid distance for determining initial clusters.
lflag	-	if on, specifies that user wants to see how isodata is progressing at various points.
tflag	-	if on, specifies that user wants a tree made of the resulting clusters.

Program Description: getparams queries the user for all parameters used in isodata that help determine how to group elements of the data set into clusters.

Algorithm: See following page

Entry: getparams (metflag,maxcy, rsplit, maxer, thetac, thetae, minel, itermax, d, lflag, tflag);

```
begin getparams;

     query user for maxcy;

     read reply;

     query user for maxmer;

     read reply;

     query user for thetac;

     read reply;

     query user for thetae;

     read reply;

     query user for mincl;

     read reply;

     query user for itermax;

     read reply;

     query user for metflag;

     read reply;

     query user for d;

     read reply;

query user if wants lflag turned on?

if reply Λ = "yes"/or Λ = "no" then do until correct reply;

     query user to reply with "yes" or "no";

end;

if reply = "yes" turn lflag on;

else turn lflag off;

query user if he wants a tree of resulting clusters;
if reply Λ = "yes"/ Λ= "no" then do till correct reply;
```

```
     query user to reply with "yes" or "no";
end;
if reply = "yes" then turn tflag on;
else turn tflag off;
end getparams;
```

Internal Subroutine Name: iclc

Calling Sequence: call iclc (tptr, dfptr, ccptr, aptr, tsptr, metflag, d, classn, incl, itnct)

Input Parameters:

tptr	-	pointer to treename file
dfptr	-	pointer to data file
ccptr	-	pointer to cluster file
aptr	-	pointer to array containing a pointer to each of the cluster files.
tsptr	-	pointer to array of temporary symbols which specify what cluster each vector is assigned to.
metflag	-	distance metric flag.
d	-	maximum centroid distance for determining initial clusters.
classn	-	number of class in treename file
inc	-	initial number of clusters to be found.
itcnt	-	number of iterations isodata has completed.

Output File Settings: A file for each cluster is created in the process directory. The name of each file is derived using the treename, nodename, "isod" and the cluster number.

Program Description: iclc generates the initial clusters by starting with the mean of the data set as the first cluster center. The distance between each vector and the cluster center(s) is calculated. If the calculated distance is greater than the user-specified distance, that vector becomes the mean of a new cluster. This cycle of calculating the distance between vectors and all cluster centers is repeated until either all vectors have been searched through or the number of clusters found is equal to the user-specified parameter (inc). Upon completion of the search for clusters, the

user is told the number of clusters found, the number of vectors examined, and asked if he wants to continue with the current number of clusters. If no, then iclc deletes all clusters and starts again.

Algorithm:

See following page

Entry: iclc (tptr, dfptr, ccptr, aptr, tsptr, metflag, d, classn, inc, itcnt)

```
begin iclc;

    get process directory;

    ask user if he wants the initial clusters generated;

    if "yes," then query user for the number of initial clusters
    to be found;

        do while user wants to loop;

            get cluster name;

            create 1st cluster file;

            put mean of data set into mean of first cluster file;

            if itcnt = 1, then initialize temporary symbols of
                vectors to zero;

            search through vectors while number of clusters is less
                than user-supplied initial number of clusters;

            call maxdis to determine distance between vector and
                cluster;

            if cluster number  = 1, then do;

                get new clustername;

                create new cluster file;

                put vector in new cluster as mean of new cluster;

            end;

        end;

    notify user of how many vectors were searched through and how many
        clusters were found.

    ask user if he wants to continue or try for more clusters;

    if he wants to try for more clusters, do;

        ask user for initial number of clusters to be found;
```

```
                delete all clusters created;
                loop through iclc again;
            end;
        end;
    end iclc;
```

Internal Subroutine Name: init

Calling Sequence: call init (lmax, lmin, tflag, metflag, theta, x)

Input Parameters:

x	-	indicates what parameters should be obtained from the user.

Output Parameters:

lmax	-	maximum link length.
lmin	-	minimum link length.
tflag	-	flag indicating user wants a MOOS tree created from the clusters.
metflag	-	distance metric flag.
theta	-	variable used in determining inconsistent links.

Program Description: init is used to obtain, from the user, all parameters needed to determine the clusters within the data set.

Algorithm: See following page.

```
Entry:                          init (lmax, lmin, tflag, metflag,
                                theta, x)

     begin init;
          query user for lmax;
          get reply;
          query user for lmin;
          get reply;

     if x=1, then do;

          query user for theta;
          get reply;
          query user for metflag;
          get reply;
          ask user if he wants a MOOS tree created from the
          clusters;

          if yes, then turn tflag on;

     end;
end init;
```

MOOS Function Name: isodata

MOOS Function Number: 217

Calling Sequence: Type in "isodata [(treename)] [(nodename)]"

Input Parameters: Standard optional data set selection parameters.

Program Description: isodata works only on the lowest node of a data tree. It takes a current data set or one supplied by the call to isodata and determines how the elements tend to group together. This is accomplished by initializing cluster centers, assigning points, computing the mean of the clusters, reassigning points and recomputing the mean until there are no changing of points between clusters. Isodata then deletes any clusters having too few points, and tries to split or merge clusters based upon a user-supplied parameter. Isodata is an iterative routine which will repeat the cycle of assigning and reassigning points any number of times specified by the user. If the user wants a tree of the resulting clusters, the file treedata is created in the process directory, with isodata then calling treeiput to create the tree. Isodata ends by calling "option".

Algorithm: See following page.

```
Entry:                    isodata [(treename)] [(nodename)]

begin isodata;

     get process directory;
     call cu-$arg-list-ptr for pointers to MOOS files
     call ut$ckparam to get current treename and nodename
     if no errors then do;
          call lnodes to check for lowest node;
          if not lowest node then do;
               notify user that isodata only operates on lowest nodes;
               terminate isodata;
          end;

          else do;
               get data file name and a pointer to the datafile;
               get pointer to treefile;
               get number of classes in treefile;
               get dimensions of vectors;

          call mainl - an internal subroutine which allocates storage
                       for a treefile structure;

          allocate Storage for temporary symbol array and array of pointers
               to cluster files;

          call getparams to get clustering parameters;

          call main2 to allocate storage for a cluster file structure;

          do while (iteration count < maximum number of iterations);

               call iclc to get initial clusters;

               call aspts to assign vectors to clusters;

               call compcent to compute the mean of each cluster;

               call covc to compute the variance and inverse covariance
                    matrices;

               call clearpts to clear out data points from cluster files;

               initialize cycle count;

               do while (cycle count <  maximum number of cycles);

                    initialize number of changes;

                    call reaspts to reassign points to clusters;
                    if any changes then do;
```

```
                call recomp to recompute mean of clusters;

                if cycle count < maximum number of cycles then
                     call clearpts to clear points from clusters;

end;

if user wants to see progress of isodata then print number
of clusters and iteration count;

end;

call dcen to delete any clusters with too few points;

if number of clusters > 2* splitting-merging parameter
     then call merger to try to merge some clusters;

else if number of clusters < 1/2* splitting-merging parameter then
     call split to try and split some clusters;

else do;

if iteration count is odd, then call split to try to split some
     clusters;

else call merger to try to merge some clusters;

end;

if there are no merges, no splits and no deletions, then set
     iteration count = maximum iterations to quit cycle;

end;

if user has indicated no tree is to be created, then do;

     ask user if he wants the current cluster files saved in the
     process directory;

     if yes, then

          rename files;

     else delete all files;

end;

if user has indicated he wants a tree of the clusters then do;

get new tree name from user;
```

```
call ctrdata to create treedata file;
call treeiput to make a MOOS tree:
end,
call option;
end isodata;
```

Internal Subroutine Name: maxdis

Calling Sequence: call maxdis (tptr, ccptr, dfptr, aptr, vn, classn, metflag, d, dist, scn)

Input Parameters:

tptr	-	pointer to treename file.
ccptr	-	pointer to cluster file.
dfptr	-	pointer to data file.
aptr	-	pointer to array containing a pointer to each cluster file.
vn	-	number of vector in data file.
classn	-	number of class in treename file.
metflag	-	distance metric flag.
d	-	maximum distance allowed between vector and cluster.

Output Parameters:

dist	-	distance between the vector, vn, and the cluster pointed to by ccptr.
scn	-	cluster number vector should be placed in.

Program Description: maxdis determines the maximum distance a vector is from all clusters. If the distance is greater than the maximum distance, d, then the vector is used to form a new cluster.

Algorithm: See following page

```
Entry:                    maxdis (tptr, ccptr, dfptr, aptr, vn,
                          classn, metflag, d, dist, scn)

    begin maxdis;

        do for each cluster;

            call dis to calculate distance between vector vn and
                 cluster pointed to by ccptr;

            if number of clusters = 1 then do;

                 if dist ≤ d then do;

                      maxdist = 0;

                   cluster number = 1;

                 end;

                  else do;

                   set maxdist = dist;

                 cluster number = number of clusters +1;

            end;

        end;

            else if number of clusters > 1 then do;

                 if n = 1 then do;

                    maxdist = dist;

                    cluster number = 1;

                 end;

                  also do;

                    maxdist = maximum of maxdist and dist;

                    if dist = maxdist and dist ≤ d then do;

                    maxdist = dist;

                    cluster number = 1;
                 end;
                 else cluster number = number of clusters + 1;
```

```
            end;
end;
end;

      dist = maxdist;

     end maxdis;
```

Internal Subroutine Name: merger

Calling Sequence: call merger (dfptr, ccptr, aptr, tsptr, metflag, nom, maxmer, thetac)

Input Parameters:

dfptr	-	pointer to data file.
ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each cluster file.
tsptr	-	pointer to array of temporary symbols which specify what cluster each vector is assigned to.
metflag	-	distance metric flag.
maxmer	-	maximum number of merges allowed.
thetac	-	maximum intercentroid distance for merging clusters.

Output Parameters

nom	-	number of merges performed.

Program Description: merger calls cdis to compute the distance between all pairs of clusters. The clusters are then sorted, in ascending order, on the distance measurement by the sort routine, sorty. If the distance measurement is less than thetac, then the two clusters are merged, and the mean of the cluster file recomputed.

Algorithm: See following page

```
Entry:                    merger (dfptr, ccptr, aptr, tsptr, metflag,
                          nom, thetac)

     begin merger;

          determine all pairs of clusters;

          compute distance between clusters of each pair;

          do a quicksort in ascending order on the distance measurement;

          do for each pair of clusters;

          if distance measurement < thetac, then do;

          add vectors of cluster 2 to cluster 1;

               reset temporary symbol of vectors in cluster 2 to value of
                    temporary symbol of cluster 1;

               recompute mean of cluster 1;

               delete cluster 2 from process directory;

               decrement number of clusters;

               increment number of merges;

               rename all cluster files;

               delete all pairs of clusters in array containing either
                    cluster 1 or cluster 2;

          end;

     end;

end merger;
```

Internal Subroutine Name: mindis

Calling Sequence: call mindis (tptr, ccptr, dfptr, aptr, vn, classn, metflag, dist, scn)

Input Parameters:

tptr	-	pointer to treename file.
ccptr	-	pointer to cluster file.
dfptr	-	pointer to data file.
aptr	-	pointer to array containing a pointer to each cluster file.
vn	-	number of vector in data file.
classn	-	number of class in treename file.
metflag	-	distance metric flag.

Output Parameters

dist	-	distance between the vector vn and the cluster pointed to by ccptr.
scn	-	number of cluster vector should be placed in.

Program Description: mindis determines which cluster the vector vn is closest to.

Algorithm: See following page.

```
Entry:                    mindis (tptr, ccptr, dfptr, aptr, vn,
                          classn, metflag, dist, scn)

     begin mindis;

          mind = large number;

           do for each cluster;

               get pointer to cluster file;

               call dis to calculate distance between
                    vector vn and cluster pointed to by ccptr;

               if dist is less than mind then do;

                  set cluster number;

                    mind = dist;

              end;

          end;

                dist = mind;

     end mindis;
```

Internal Subroutine Name: mindisl

Calling Sequence: call mindisl (ccptr, dfptr, aptr, vn, metflag, dist, scn)

Input Parameters:

ccptr	-	pointer to cluster file.
dfptr	-	pointer to data file.
aptr	-	pointer to array containing a pointer to each cluster file.
vn	-	number of vector in data file.
metflag	-	distance metric flag.

Output Parameters:

dist	-	distance between vector vn and the cluster pointed to by ccptr.
scn	-	number of cluster vector should be placed in.

Program Description: mindisl determines which cluster the vector vn is closest to.

Algorithm: See following page

Entry: mindisl (ccptr, dfptr, aptr, vn, metflag, dist, scn)

```
begin mindisl;

     mind = some large number;

     do for each cluster;

          get pointer to cluster file;

          call disl to calculate distance

               between vector vn and cluster pointed to
               by ccptr;

          if dist < mind then do;

               set cluster number;

               mind = dist;

          end;

     end;

     dist = mind;

     end mindisl;
```

MOOS Function Name: mst

MOOS Function Number: 218

Calling Sequence: Type in "mst [(treename)][(nodename)]"

Input Parameters: standard optional data set selection parameters.

Program Description: mst works only on the lowest node of a data set. It takes either a current data set or one supplied by the call to mst, and detects elongated clusters within the data set. This is accomplished by linking all elements in the data together as one cluster, and then deleting any links that are either too short or too long, as specified by the user. The user has the option of changing the link length values at the completion of link deletion. Mst will repeat the cycle of deleting links if the length value is changed. If the user wants a tree of the resulting clusters, the file treedata is created in the process directory, with mst then calling treeiput to create the tree. Mst ends by calling "option".

Algorithm: See following page

```
Entry:                          mst [(treename)] [(nodename)]

     begin mst;

          get process directory;

          call cu_$arg_list_ptr for pointers to MOOS files;

          call ut$ckparam to get current treename and nodename;

          if no errors then do;

               call lnodes to check for lowest node;

               if not lowest node then do;

                    notify user that mst only operates on lowest nodes;

                    terminate mst;

               end;

               else do;

                  get data file name and a pointer to the datafile;

                  get pointer to treefile;

                  get dimensions of vectors;

                  call main1 - an internal subroutine which allocates
                               storage for a treefile structure;

                  allocate storage for c&p array and array of pointers
                    to cluster files;


                  call init to get clustering parameters;

                  call main2 to allocate storage for a cluster file
                    structure;

                  call trip to create triple array of all pairs of
                    data points & the distance between the elements of
                    the pair;

                  call sorty to do a quicksort on the distance element
                    of the triple array;

                  zero out c-array;
```

```
                    call flinks to find all links and create both
                      the c and p array;

                    do while (user wishes);

                      initialize all elements of c-array to 1;
                      create first cluster file;

                    call dlink to delete any inconsistent links
                      (i.e. too long or too short);

                    call pcfile to put elements of data set in
                      correct cluster file;

                    ask user if he wants to change link length values;

                    if yes, then do;

                      delete all clusters;

                      call init to get new link length values;

               end;

                else quit loop;

          end;

if user has indicated no tree is to be created then do;

     ask user if wants the current cluster files saved in the process
     directory;

     if yes then rename files;

     else delete all files;

end;

     if user has indicated he wants a tree of the clusters then do;

     get new tree name from user;

     get total number of vectors in all clusters;

     call ctrdata to create treedata file;

     call treeiput to make a MOOS tree;

     end;

     call option;

     end mat;
```

Internal Subroutine Name: pcfile

Calling Sequence: call pcfile(dfptr, aptr, ccptr, metflag, cptr)

Input Parameters:

dfptr	-	pointer to data file.
aptr	-	pointer to array containing a pointer to each cluster file.
ccptr	-	pointer to cluster file.
metflag	-	distance metric flag.
cptr	-	pointer to c array.

Program Description: pcfile searches through the data set and places each vector into the cluster it has been assigned to. The mean and variance or covariance of each vector is then calculated. The user is notified of the number of clusters found and the number of points in each cluster.

Algorithm:

Entry: pcfile(dfptr, aptr, ccptr, metflag, cptr)

```
begin pcfile;

     do 1 to number of clusters;

          get pointer to file;

          initialize number of points to zero;

          search through vectors and place in correct cluster;

     end;

     call compcent to calculate the mean of each cluster;

     if metflag ^= "e" then call covc to calculate the variance or
     inverse covariance of cluster;

     print the number of clusters and the number of points in each
     cluster;

end pcfile;
```

Internal Subroutine Name: reaspts

Calling Sequence: call reaspts (dfptr, ccptr, aptr, tsptr, metflag, tnoc, classn)

Input Parameters:

dfptr	-	pointer to data file.
ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each cluster file.
tsptr	-	pointer to array of temporary symbols which indicate what cluster each vector is assigned to.
metflag	-	distance metric flag.
classn	-	number of class in treename file.

Output Parameter:

tnoc	-	counter which indicates the number of vectors transferring from one cluster to another.

Program Description: reaspts examines each vector whose temporary symbol is not zero, calls mindisl to determine which cluster the vector is closest to, and assigns the vector to that cluster. If the user has specified to see the progress of isodata, then the number of points in each cluster is printed.

Algorithm: See following page

Entry: reaspts (dfptr, ccptr, aptr, tsptr, metflag, tnoc, classn)

```
     begin reaspts;

          do for each vector;

          if temporary symbol of vector ^= zero, then do;

          call mindisl to determine which cluster the vector is
               closest to;

          if the temporary symbol ^= the new cluster number then reset
               temporary symbol to a negative number;

          get pointer to new cluster;

          increment number of points in cluster;

          add vector to cluster file;

          end;

end;

     If list flag is on, then do;

     print the number of vectors which changed clusters;

     print number of vectors in each cluster;

     end;

end reaspts;
```

Internal Subroutine Name: recomp

Calling Sequence: call recomp (ccptr, aptr, tsptr)

Input Parameters:

ccptr	-	pointer to cluster file.
aptr	-	pointer to array containing a pointer to each cluster file.
tsptr	-	pointer to array of temporary symbols which specify what cluster each vector is assigned to.

Program Description: recomp examines the temporary symbols of all vectors in the data file. A negative temporary symbol indicates the vector changed clusters, requiring the mean for both the old and the new cluster to be recomputed.

Algorithm: See following page

```
Entry:                          recomp (ccptr, aptr, tsptr)
     begin recomp;
          do for each vector;
               if temporary symbol is negative, then do;
                    get old cluster number;
                    get new cluster number;
                    get pointer to old cluster;
                    initialize mean to zero;
                    if number of points Λ = 0, then compute mean;
                    get pointer to new cluster;
                    initialize mean to zero;
                    if number of points Λ = 0, then compute mean;
               end;
          end;
end recomp;
```

Internal Subroutine Name: sorty

Calling Sequence: call sorty(trptr)

Input Parameters:

trptr - pointer to structure being sorted.

Program Description: sorty uses a quicksort to sort an array containing all pairs of vectors of the data set and the distance between each vector of the pairs. The sort operates on the distance element and produces the same array in ascending order.

Algorithm:

Entry: sorty(trptr)

```
  begin sorty;
       determine software stack size and allocate storage for it;
       if number of elements to be sorted is > 9 then do;
            initialize stack pointer;
            initialize maximum boundaries;
            do loop;
                 initialize left and right boundaries;
                 get key and record to be sorted;
       Q3:       do while (go);
                 increment left boundary;
                 if (distance of left boundary < key), then go = no;
                 if left boundary = maximum left boundary then go = no;
                 end;
            do while (go);
                 decrement right boundary;
```

```
     if key < right boundary then go = no;

     if right boundary = 1, then go = no;

end;

if right boundary > left boundary, then do;

     exchange records;

     if maximum right boundary - right boundary ≥ right
     boundary - left boundary > 9, then do;

          increment stack pointer;

          decrement maximum right boundary;

          set stack;

     end;

     else if (right boundary - left boundary > max right
     boundary - right boundary > 9), then do;

          increment stack pointer;

          set stack;

          set maximum left boundary;

     end;

     else if max right boundary - right boundary > 9 ≥ right
     boundary - left boundary, then reset max left boundary;

     else if right boundary - left boundary > 9 ≥ max right
     boundary - right boundary, then reset max right boundary;

     else if stack pointer ^= zero, then do;

          reset maximum left and right boundaries;

          decrement stack pointer;

     end;

     else if stack pointer = zero then quit loop;

end;
```

```
          else do;
               exchange records;
               go to Q3;
          end;
     end;
end;
do j = 2 to end of array;
          if key (j - 1) > key (j) then do;
               i = j - 1;
               do while (go);
                    if i = 0, then do;
                         if key(i) > key(j), then do;
                              let Record (i+1) = Record (i);
                              decrement i;
                         end;
                         else go = no;
                    end;
                    else go = no;
               end;
               Record (i+1) = Record (j);
          end;
     end;
     end sorty;
```

Internal Subroutine Name: split

Calling Sequence: call split (tptr, dfptr, ccptr, aptr, tsptr, metflag, nos, thetae, nsplit, classn)

Input Parameters:

- tptr - pointer to treename file.
- dfptr - pointer to data file.
- ccptr - pointer to cluster file.
- aptr - pointer to array containing a pointer to each cluster file.
- tsptr - pointer to array of temporary symbols which specify what cluster each vector is assigned to.
- metflag - distance metric flag.
- thetae - used in determining when to split a cluster.
- nsplit - used in determining whether to split or merge a cluster.
- classn - number of data class in treename file.

Output Parameters:

- nos - number of splits performed.

Program Description: split examines each cluster and tentatively chooses it for splitting if the cluster standard deviation for any dimension exceeds the product of thetae and the standard deviation of that dimension in the original data set. The vectors are then assigned to clusters according to whether they are above or below the cluster mean on the dimension causing the split. The mean of each cluster is calculated and if the distance between the two clusters is at least 1.1 times thetae, then the split is accepted.

Algorithm: See following page

```
Entry:                    split (tptr, dfptr, ccptr, aptr, tsptr,
                          metflag, nos, thetae, nsplit, classn)

     begin split;

          calculate the standard deviation of each cluster;

          get the standard deviation of the original data set;

          do i = 1 to number of clusters;

               get pointer to cluster file;

               if cluster standard deviation for any dimension exceeds
                    thetae*  the standard deviation of the original data
                    set on the same dimension, then do;

                    increment number of clusters;

                    create new cluster file;

                    assign vectors to clusters - those above the mean on the
                    splitting dimension are put in the new cluster - those
                    below the mean on the splitting dimension are left in
                    the old cluster;

                    calculate new means of both clusters;

                    call cdis to compute distance between the two clusters;

               if distance is < 1.1 times thetae, then split is unacceptable
               and do;

                    put all vectors back in original cluster;

                    delete new cluster;

                    decrement number of clusters;

          end;

          else split is acceptable and do;

               increment number of splits;

               reset temporary symbols of split cluster;

               end;

          end;

     end;

end split;
```

Internal Subroutine Name: trip

Calling Sequence: call trip (tptr, dfptr, trptr, metflag, classn)

Input Parameters:

tptr	-	pointer to treename file.
dfptr	-	pointer to data file.
trptr	-	pointer to tarray.
metflag	-	distance metric flag.
classn	-	number of class in treename file.

Output File Settings: creates the file "tripfile" in the process directory.

Program Description: trip creates the file "tripfile" which contains all pairs of data elements in the original data set and the distance between the two elements of each pair.

Algorithm: See following page.

Entry: trip (tptr, dfptr, trptr, metflag classn)

```
begin trip;

     determine size of tarray based on number of vectors;
     allocate storage for tarray;
     create trip file;
          do for all pairs of data elements;
               call vdis to determine distance
                    between the two elements;
               store elements and distance in
                    trip file;
          end;
     end trip;
```

Internal Subroutine Name: vdis

Calling Sequence: call vdis (dfptr, tptr, classn, v1, v2, metflag, dist)

Input Parameters:

dfptr	-	pointer to data file.
tptr	-	pointer to treename file.
classn	-	number of class in treename file.
v1	-	vector 1.
v2	-	vector 2.
metflag	-	distance metric flag.

Output Parameters:

dist	-	distance between v1 and v2.

Program Description: vdis calculates the distance between the two vectors, v1 and v2. The distance metric used is specified by the metflag.

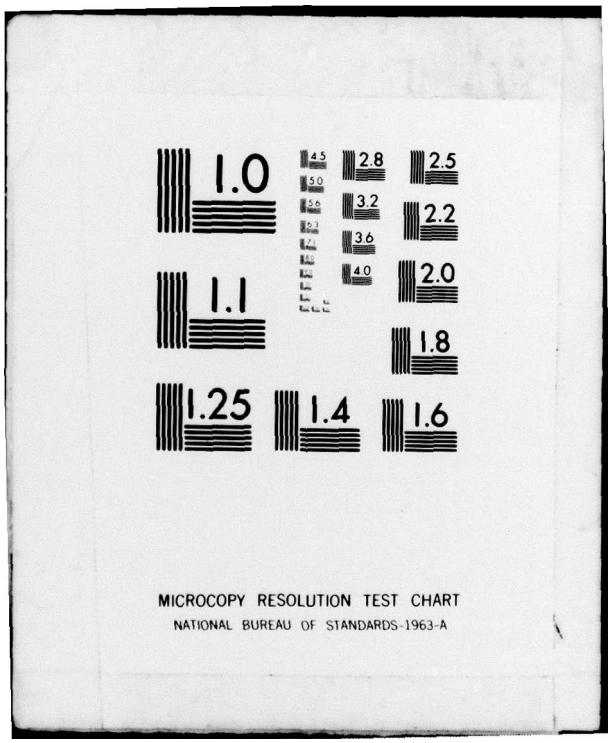
Algorithm: See following page

AD-A080 626 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS REMOTE DATA ENTRY SYSTEM. VOLUME II. CLUSTERING ADDITIO--ETC(U)
OCT 79 J D DYAR F30602-77-C-0174
UNCLASSIFIED PAR-79-53 RADC-TR-79-265-VOL-2 NL

2 OF 2
AD
A080626



END
DATE
FILMED
3 - 80
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
Entry:                          vdis (dfptr, tptr, classn, v1,
                                v2, metflag, dist)

     begin vdis;

          if metflag = "e" then use euclidean distance formula;

     else if metflag = "w" then use weighted vector formula;

     else if metflag = "m" then use mahalanobis distance formula;
          calculate distance;

end vdis;
```

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.